Applications Performance Under OSF1/AD and SUNMOS on Intel Paragon XP/S-15

Subhash Saini and Horst D. Simon[1]
Report RNR-94-016, August 1994

NAS Systems Division
Applied Research Branch
NASA Ames Research Center, Mail Stop T27A-1
Moffett Field, CA 94035

Abstract. On the Paragon, two operating systems are available: OSF/1 AD, and SUNMOS. The chief drawbacks of OSF/1 AD are that OSF/1 AD takes about 8 MB of memory on each node of the Paragon, and that messages can be sent only at a bandwidth of 30-35 MB per second compared to 200 MB per second peak advertised rate. Furthermore latencies are on the order of 100 microseconds using Intel NX calls under OSF1/ AD. All these drawbacks can be minimized by using SUNMOS. SUNMOS takes only 250 KB of memory on each node and can send messages at a bandwidth of 170 MB per second with latencies of 70 microseconds. We have measured the performance of applications under OSF/1 AD and SUNMOS and found that under OSF/1 AD, performance does not scale as the number of nodes increases, whereas under SUNMOS it seems to scale because of higher communication bandwidth.

This report presents the state of Paragon operating systems as of Spring 1994. Specifically, it compares OSF release R1.1 and SUNMOS S1.0. Later releases of both operating systems have addressed some of the shortcomings mentioned here.

To appear in the Proceedings of Supercomputing '94.

[1]Both authors are employees of Computer Sciences Corporation. This work was supported through NASA Contract NAS 2-12961.

Applications Performance Under OSF/1 AD and SUNMOS on Intel Paragon XP/S-15*

Subhash Saini and Horst D. Simon
Computer Sciences Corporation at

NASA Ames Research Center, Moffett Field, CA 94035-1000

Abstract

On Paragon, two operating systems are available: (a) OSF/1 AD, and (b) SUNMOS. The chief drawbacks of OSF/1 AD are (a) OSF/1 AD takes about 8 MB of memory on each node of the Paragon, (b) messages can be sent only at a bandwidth of 30-35 MB per second compared to 200 MB per second peak advertised rate, (c) latencies are on the order of 100 microseconds using Intel NX calls under OSF1/ AD. All these drawbacks can be minimized by using SUNMOS. SUNMOS takes only 250 KB of memory on each node and can send messages at bandwidth of 170 MB per second with latencies of 70 microseconds. We have measured the performance of applications under OSF/1 AD and SUNMOS and found that under OSF/1 AD, performance does not scale as the number of nodes increases, whereas under SUNMOS it seems to scale because of higher communication bandwidth.

1: Introduction

The Numerical Aerodynmical Simulation (NAS) Systems Division received an *Intel Touchstone Sigma* prototype model *Paragon XP/S-15* in February 1993. It was found that performance of many applications including the assembly coded single node *BLAS 3* routine DGEMM [1] was lower than the performance on *Intel iPSC/860*. This finding was quite puzzling since the clock of the microprocessor *i860 XP* used in the *Paragon* is *25%* faster than the microprocessor *i860 XR* used in the *Intel iPSC/860* [2]. It was also found that the performance of the *NAS Parallel Benchmarks (NPB)* [3-5] is enhanced by about *30%* if they are run for second time in a DO loop. Furthermore, the performance of DGEMM was identical for the first run and the second run on a service node, but on a compute node the performance of the second run was about *40%* better than the first run. These anomalies in the performance on the *Paragon* led us to investigate the problem in more detail. This, in turn, led us to propose a method of dynamic allocation of memory that increases the performance of the applications by about *30%* to *40%* [15-16].

By November 1993, it was realized that the performance of the applications is very much limited by two mains factors (a) high latency (120 microseconds), and (b) low communications bandwidth (30 - 35 MB per second). The optimal use of the Paragon was also limited by the fact that the micro kernel and Open Software Foundation (OSF)/1 AD [6-7] server takes 8 MB per node, thereby leaving only 8 MB for the user application. On the other hand, there is another operating system available for the Paragon, tailored to the needs of applications requiring a large volume of communication: the Sandia University of New Mexico operating system (SUNMOS, [8-9]). The latency under SUNMOS is 70 microseconds and bandwidth is 170 MB per second. Also, SUNMOS needs only 250 KB of memory per node, thereby making 15.8 MB of memory available for the user application. Additionally, there is no degradation in performance associated with using static allocation of memory. In view of this, it was decided to port, test and evaluate the performance of SUNMOS on the NAS Paragon. In January 1994, memory per node was upgraded from 16 MB to 32 MB.

In Section 2 we give a brief overview of the Paragon system. Section 3 gives some details on operating systems available on Paragon. Section 4 gives the description of the applications we have studied under OSF/1 AD and SUNMOS. Section 5 describes the methodology used. Section 6 presents results and discussion. Lastly, Section 7 contains the conclusions of the paper.

2: Overview of the Paragon

2.1: The i860 XP microprocessor

The *Paragon* system is based on the *64* bit *i860 XP*^{TM} microprocessor [2] by Intel. The *i860 XP*^{TM} microprocessor has *2.5* million transistors in a single chip and runs at *50 MHz*. The theoretical speed

is *100 MFLOPS* in *32* bit floating point and *75 MFLOPS* for *64* bit for floating point operations. The *i860 XP*^{TM} features *32* integer address registers with *32* bits each. It has *32* floating point registers with *32* bits each. The floating point registers can also be accessed as *16* floating point registers with *64* bits each or *8* floating point registers with *128* bits each. Each floating point register has two read ports, a write port and two-bidirectional ports. All these ports are *64* bits wide and can be used simultaneously. The floating point registers serve as input to the floating point adder and multiplier. In vector computations, these registers are used as buffers while the data cache serves as vector registers. The *i860 XP*^{TM} microprocessor has *16 KB* of instruction cache and *16 KB* of data caches. The data cache has a *32* bit path to the integer unit and *128* bit data path to the floating point unit. The *i860 XP*^{TM} has a number of advanced features to facilitate high execution rates. The *i860 XP*^{TM} microprocessor's floating point unit integrates single-cycle operation, *64* bit and *128* bit data paths on chip and a *128* bit data path to main memory for fast access to data and transfer of results. Floating point add, multiply and fetch from main memory are pipelined operations, and they take advantage of a three-stage pipeline to produce one result every clock for *32* bit add or multiply operations and *64* bit adds. The *64* bit multiplication takes two clocks.

2.2: NAS Intel Paragon XP/S-15

A single node of the *Paragon XP/S-15* [10] consists of two *i860 XP*^{TM} microprocessors: one for computation and the other for communication. The compute processor is for computation and the communication processor handles all message-protocol processing thus freeing the computation processor to do computations. Currently, the communication processor is *not* used in the *NAS Paragon*. Each compute processor has 32 *MB* of local memory but at *NAS* only about 24 *MB* is available for applications, the rest being used for the micro kernel, *OSF* server and system buffers.

The *NAS Paragon* has *256* slots for nodes. Slots are given physical node numbers from *0* through *255*. Slots are physically arranged in a rectangular grid of size *16* by *16*. There are *8* service nodes; four of them have *16 MB* of memory each and the other four have *32 MB* of memory each. Column *0* and column *14* have no physical nodes. The service partition contains *8* nodes in the last column. One of these service nodes is a boot node. This boot node has *32 MB* of memory and is connected to a *Redundant Array of Independent Disks-1 (RAID-1)*. The compute partition has *208* nodes which occupy columns *1* through *13*. Compute processors are given logical numbers *0* through *207*. Compute processors are arranged in a *16* by *13* rectangular grid. The *227* nodes are arranged in a two-dimensional mesh using wormhole routing network technology. The four service nodes comprise the service partition and provide an interface to the outside world, serving as a *front end* to the *Paragon* system. Besides running jobs on the compute nodes, the service nodes run interactive jobs, such as *shells* and *editors*. They appear as one computer running *UNIX*.

Theoretical peak performance for *64* bit floating point arithmetic is *15.6 GFLOPS* for the *208* compute nodes. Hardware node-to-node bandwidth is *200 MB* per second in full duplex.

The nodes of the *NAS Paragon* are organized into groups called partitions [10]. Partitions are organized in a hierarchical structure similar to that of the *UNIX* file system. Each partition has a *pathname* in which successive levels of the tree are separated by a periods ("."), analogous to "/" in the *UNIX* file system. A subpartition contains a subset of the nodes of the parent partition.

Currently, on the *NAS Paragon* there are no subpartitions of *.compute* or *.service*. The root partition (denoted by ".") contains all *227* nodes of the *Paragon*. There are two subpartitions of the root partition: the compute partition, named *.compute*, contains *208* nodes to run parallel applications. The service partition, named *.service*, contains four nodes devoted to interactive jobs. The remaining eight nodes are not part of a subpartition and serve as disk controllers and are connected to the *RAID* for *I/O*. The four nodes of the service partition appear as one computer. In summary, the *NAS Paragon* system has *208* compute nodes, 3 *HiPPI* nodes, *1* boot node, *8* disk nodes, *4* service nodes of which *1* is a boot node and *4* nodes are not used at this time, for a total of *227* nodes. When a user logs onto the *Paragon*, the *shell* runs on one of the four service nodes. In the current release of the *Paragon OS*, processes do not move between service nodes to provide load balancing. However, the load leveler decides on which node a process should be started. In principle, partitions and subpartitons may overlap. For instance, there could be a subpartition called *.compute.part1* consisting of nodes *0-31* of *.compute*, and another subpartition called *.compute.part2* consisting of nodes *15-63* of *.compute*. However, in the current release of the

operating system on the *NAS Paragon*, there are two problems which restrict the use of subpartitions. First, running more than one application on a node (either two jobs in the same partition or jobs in overlapping partitions) may cause the system to crash. Second, the existence of overlapping partitions sometimes causes jobs to wait when they need not. For these two reasons, there are currently no subpartitions of the *.compute* partition. All jobs run directly on the *.compute* partition.

3: Operating Systems on Paragon

On NAS Paragon the following two operating systems are available.

3.1: Open Software Foundation (OSF/1 AD)

The *UNIX* operating system was originally designed for sequential computers and is not very well suited to the performance of massively parallel applications. The *Paragon* operating system is based upon two operating systems: the *Mach* system from *Carnegie Mellon University* and the *Open Software Foundation's OSF/1 AD* distributed system for multicomputers [7,8]. The *Paragon's* operating system provides all the *UNIX* features including *virtual memory*; *shell*, *commands* and *utilities*; *I/O* services; and networking support for *ftp*, *rpc* and *NFS*. Each *Paragon* node has a small microkernel irrespective of the role of the node in the system. The *Paragon* operating system provides programming flexibility through virtual memory. In theory, virtual memory simplifies application development and porting by enabling code requiring large memory to run on a single compute node before being distributed across multiple nodes. The application runs in virtual memory which means that each process can access more memory than is physically available on each node. At NAS, OSF/1 AD runs on 144 compute nodes and on all service nodes.

The *Paragon OS* used in this study is version *R1.1.* and the *Fortran* compiler is *4.1* [10]. The compiler options used are the f77 -O4 -Mvect -Knoieee abc.f -lkmath and the compilation was done on the service node. There is a compiler option by which one may set the size of the portion of the cache used by the vectorizer to *number* bytes. This *number* must be a multiple of *16*, and less than the cache size 16384 of the microprocessor *i860 XP*. In most cases the best results occur when *number* is set to *4096*, which is the default. In view of this we decided to choose the default size *4 KB* and the highest optimization level of *4* was used. This level of optimization generates a basic block for each *Fortran* statement and scheduling within the basic block is performed. It does perform aggressive register allocation for software pipelined loops. In addition, code for pipelined loops is scheduled several ways, with the best way selected for the assembly file. The option -Knoieee was used, which produces a program that flushes denormals to *0* on creation (which reduces underflow traps) and links in a math library that is not as accurate as the standard library, but offers greater performance. This library offers little or no support for exceptional data types such as *INF* and *NaN*, and will not trap on such values when encountered. If used while compiling, it tells the compiler to perform real and double precision divides using an in-line divide algorithm that offers greater performance than the standard algorithm. This algorithm produces results that differ from the results specified by the *IEEE* standard by no more than three units in the last place *(ulp)*.

3.2: SUNMOS

The chief drawbacks of OSF/1 AD are (a) OSF/1 AD takes about 8 MB of memory on each node of the Paragon, (b) messages can be sent at a bandwidth of 30-35 MB per second compared to 200 MB per second peak advertised rate, (c) latencies are of the order of 120 microseconds using Intel NX calls under OSF/1 AD [6, 7]. All these drawbacks can be minimized by using a new operating system called Sandia University of New Mexico Operating System (SUNMOS) [8, 9]. SUNMOS was originally developed and ported to nCUBE-2 in 1991. SUNMOS was ported to Intel Paragon in 1993. SUNMOS takes only 250 KB of memory on each node of the Paragon and can send messages at bandwidth of 170 MB per second with latencies of 70 microseconds. However, SUNMOS does not provide a complete implementation of Intel's NX message-passing library. It only supports hostless programs and does not support host-node programs. In addition to these limitations, SUNMOS has very limited support for I/O and absolutely no support for parallel I/O. NAS Paragon runs SUNMOS on 64 compute nodes and these nodes do not appear under the compute partition. Furthermore, jobs running under SUNMOS get a fixed amount of heap, stack, and communication space at load time. SUNMOS does not support virtual memory Figure 1 shows the typical installation of SUNMOS on Paragon. SUNMOS runs only on the compute nodes. Comparative performance of operating systems OSF/1 AD and SUNMOS is given in Table

1. It is possible to run SUNMOS either on all the compute nodes or on a subset of them. One may not notice any difference when SUNMOS is running on part of compute nodes and OSF/1 AD running on the remaining compute nodes other than the fact that there are fewer compute nodes in the compute partition. SUNMOS never runs on the service nodes of the service partition.

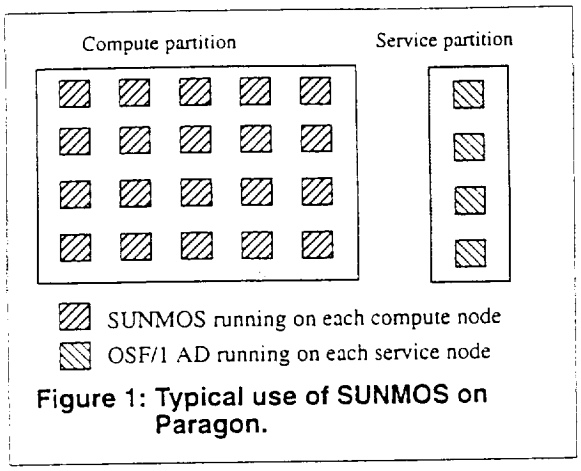

Figure 1: Typical use of SUNMOS on Paragon.

3.2.1: The utility YOD:

When SUNMOS runs on part of the compute nodes, a program called *yod* also runs on the service node/nodes allocated to compute partition running SUNMOS. It is illustrated in Figure 2. The utility *yod* [8] is used to allocate a partition of the SUNMOS to a portion of the mesh and load the executable. This utility runs in the service partition and handles all requests from the SUNMOS compute nodes that it controls. Aborting a job under control of *yod* by KILL -9 leaves the nodes allocated and pr the use of these nodes in subsequent runs until the system is rebooted. Among others, there are three arguments to *yod* which need careful attention. These are *comm*, *heap* and *stack*. The argument *stack* reserves the space for stack. In many examples studied, inadequate allocation of stack gave run time errors and the benchmark could not be run. The switch *comm* sets aside space that is used for buffering messages for which no receive has been posted. The default size for *comm* is 256 KB. If the communication buffer overflows during execution of the application it causes an unrecoverable error and sometimes the system hangs and needs to be rebooted. The argument *heap* reserves the space for heap. The default is to allocate the remaining memory left on each node after the *comm*, *stack*, program (text and data) and OS space have been allocated. Currently, the size of the heap cannot be more than 16 MB. When the application needs more than 16 MB per node, specifying the heap to be 16 MB will not run the application and gives the message that not enough space is available for running the application and sometimes the system hangs and needs to be

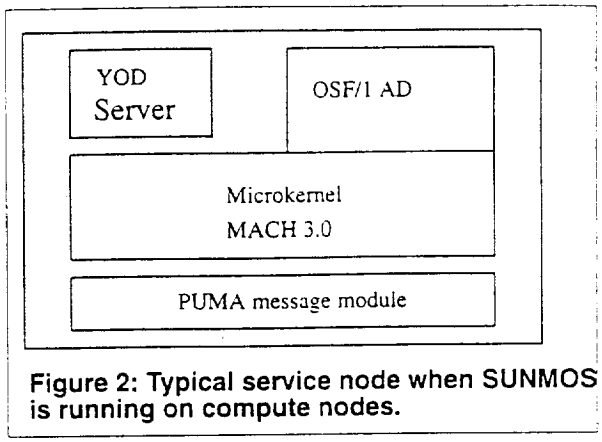

Figure 2: Typical service node when SUNMOS is running on compute nodes.

rebooted. The argument *heap* reserves the space for heap. The default is to allocate the remaining memory left on each node after the *comm*, *stack*, program (text and data) and OS space have been allocated. Currently, the size of the heap cannot be more than 16 MB. When the application needs more than 16 MB per node, specifying the heap to be 16 MB will not run the application and gives the message that not enough space is available for running the application. The application can be

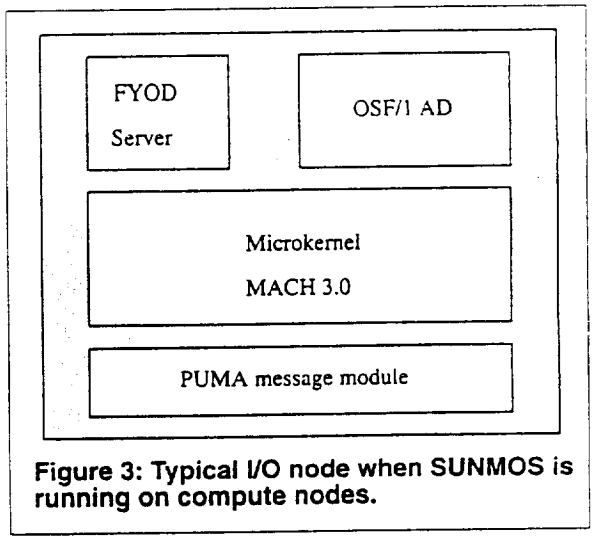

Figure 3: Typical I/O node when SUNMOS is running on compute nodes.

still run up to 25 MB per node if the *heap* argument is not used. Occasionally, the application hangs or the system crashes. The application always hangs the system or crashes it if the application needs more than 25 MB per node. To run

Table 1: Comparative performance of operating systems OSF/1 AD and SUNMOS.

Features	Operating System	
	OSF/1 AD	SUNMOS
Microkernel	5 MB	0.25 MB
Server	3 MB on compute node	None
Memory for application	24 MB per node	31.75 MB per node
Virtual memory	Yes	No
Latency	120 micro seconds	70 microseconds
Bandwidth	35 MB per second	170 MB per second
I/O - READ	8 MB per second	2 MB per second
I/O - WRITE	11 MB per second	3 MB per second
Support	Intel	Sandia Nat. Laboratories
Technology	Mixture of OSF, LOCUS and Intel	Sandia National Laboratories and Univ. of New Mexico
Reliability	3 crashes per day	Undetermined
Allocation of heap and stack	By OSF/1 AD	By the user
Time sharing	Yes	No
Nodes on which runs	Compute & Service nodes	Only on compute nodes
Parallel file system	Yes	No
Availability of debugger	Yes	No
Reliability of results	High	Intermittently wrong results if more than 16 MB per node is used
Functionality	High	Low
Memory bandwidth	Low	High
Scalability	Low	High
Applications performance	Low	Moderate

the application successfully one has to be extra careful in allocating heap and a stack.

3.2.2 The utility FYOD:

fyod is a utility that starts a SUNMOS file server [8]. Typically, it runs on an I/O node with a disk attached to it in the service partition. All input/output for an application is routed through *yod* utility. The purpose of *fyod* is to remove this bottleneck and distributes the work load among several I/O nodes in the service partition. The use of *fyod* is transparent to the user. It improves the performance of applications that write to many files simultaneously

SUNMOS can run under three modes:
(a) **mode 0:** Second processor is not used.
(b) **mode 1:** Second processor is used as a communication processor.
(c) **mode 2:** Second processor is used for computation.

4: Applications used

4.1: BLAS

BLAS 1, 2 and *3* are the basic building blocks for many scientific and engineering applications. For example, the dot product (BLAS 1) is a basic kernel in *Intel's ProSolver Skyline Equation Solver (ProSolver-SES)* [11], a direct solver using skyline storage, useful for performing Finite Element Structural analysis in designing aerospace structures. *BLAS 3 (matrix-matrix)* kernels are basic kernels in *Intel's ProSolver Dense Equation Solver (ProSolver-DES)* [12], a direct solver that may be applied in solving computational electromagnetics *(CEM)* problems using *Method of Moments (MOM). BLAS 2* and *BLAS 3* are basic kernels in *LAPACK* [1]. In the present paper, we have used a *BLAS 3* routine called DGEMM to compute $C = A*B$, where A and B are real general matrices. The DGEMM is a single node assembly coded routine [17] and as such involves no interprocessor communication

4.2: Fast Fourier Transforms

The FFT is a basic tool in various scientific and engineering applications ranging from artificial intelligence to oil exploration. At NAS, distributed three-dimensional FFT is used to solve the Poisson partial differential equation. We have measured the performance of radix-2, -3 and -5 complex to complex FFT on Paragon. We have used (a) 1-D single node, and (b) 3-D distributed FFTs to study the impact of data cache usage and scalability issues under OSF and SUNMOS on Paragon. Intel supplies only radix-2 1-D FFTs on Paragon [17].

4.3: NAS Parallel Benchmarks

The *NPB* [3-5] were developed to evaluate the performance of highly parallel supercomputers. One of the main features of these benchmarks is their *pencil and paper* specification, which means that all details are specified algorithmically, thereby avoiding many of the difficulties associated with traditional approaches to evaluating highly parallel supercomputers. The *NPB* consist of a set of eight problems each focusing on some important aspect of highly parallel supercomputing for computational aerosciences. The eight problems include five kernels and three simulated computational fluid dynamics *(CFD)* applications. The implementation of the kernels is relatively simple and straightforward and gives some insight into the general level of performance that can be expected for a given highly parallel machine. The other three simulated *CFD* applications need more effort to implement on highly parallel computers and are representative of the types of actual data movement and computation needed for computational aerosciences. The *NPB* all involve significant interprocessor communication with the exception of the Embarrassingly Parallel *(EP)* benchmark which involves almost no interprocessor communication.

4.4: NAS Kernels

This set of computational kernels comes from applications at NASA Ames. It demonstrates the compiler's ability to vectorize floating point operations as well as processor speed [14].

5: Procedure for 1st Run and 2nd Run

It was found that the performance of *NPB* codes is enhanced by about *30%* if they are run for a second time in a DO loop. Furthermore, the performance of DGEMM was identical for the first run and second run on a service node but on a compute node the performance of the second run was about *40%* better than the first run. In our numerical results section we will present results for a first run and a second run of each application. The procedure to obtain first run and second run for a given application is illustrated in Table 2. In this

table, a DO loop index i running from *1* to *2* is inserted just before the section of the code we want to time for benchmark purposes. In this table the *first run* corresponds to *i=1* and the *second run* corresponds to *i=2* as shown in Table 2. The overhead in calling the function DCLOCK was estimated to be about $1.5x10^{-6}$second [13, 15-16].

Table 2: Procedure for obtaining first run and second run.

```
PROGRAM abc
...
DO i = 1, 2
t0 = DCLOCK()
t1 = DCLOCK
CALL DGEMM( ,..., ...)
t2 = DCLOCK()
time = t2 - (t1 - t0)
ENDDO
...
END
```

6: Results and Discussion

The results were obtained under OSF/1 AD 1.1 and SUNMOS S1.1. Figure 4(a) shows the results for the assembly coded BLAS 3 routine DGEMM for a matrix size of 1024x1024 on one compute node for the first and second run. The performance is 27 MFLOPS for the first run and 46 MFLOPS for the second run.

The performance obtained by the second run is about 40% better than the performance obtained by the first run. This degradation in performance is not acceptable since users will always run their code once. The left of the Figure 4(b) shows the performance of DGEMM on four compute nodes. The MFLOPS rate has decreased from 27 MFLOPS to about 6 MFLOPS. This problem can be eliminated by using dynamic allocation of memory. The performance of DGEMM using dynamic allocation of memory is shown on the right side of Figure 4(b). For details see reference [13, 15-16]. The performance of DGEMM as a function of the size of the matrix is shown in Figure 5. SUNMOS results are about 5% better than corresponding OSF results probably due to a better memory access mechanism. However, under SUNMOS, for a matrix size larger than 1024 the system either hangs or crashes and, therefore, results could not be obtained. Under OSF we could run our matrix beyond 32 MB per node but with 40% decrease in performance.

Figure 6(a) shows the performance of assembly coded 1-D FFT [17]. Performance is much better when twiddle factors are in cache. The performance is a maximum at x=512, corresponding to a of 4 KB data cache size. The other peaks reflect higher harmonics at multiple of 512. Figures 7(a) and 7(b) are similar to Figure 6 but are for Fortran coded 1-D FFT. Notice that the performance does not decrease after x=512 but remains constant due to better cache management. Also Fortran coded 1-D FFT is for radix-2, -3 and -5 whereas assembly coded 1-D FFT supplied by the Intel and used in Figure 3 is radix-2. Figure 8(a) shows the performance of NPB under OSF for 128 nodes. Performance is under 1 GFLOPS except for benchmarks EP, FT and BT. The benchmark EP involves almost no communication. The kernel FT uses assembly coded 1-D FFT and BT uses an assembly coded block tridiagonal solver for most of the computations. Figure 8(b) shows the performance in MFLOPS per node for NPB. The average performance is about 5 MFLOPS except for EP, FT and BT for the same reasons as discussed before. High performance of FT (9 MFLOPS) and BT (10 MFLOPS) is due to the use of assembly coded routines.

Figure 9(a) shows the performance in MFLOPS of 3-D FFT as a function of the number of compute nodes for a fixed problem size of *256x256x128*. Under OSF, performance does not scale, whereas under SUNMOS it seems to scale because of higher communication bandwidth. Figure 9(b) shows the performance of 3-D FFT as a function of the size of the FFT for a fixed system size (128 nodes) for OSF and SUNMOS. Here also the scalability under SUNMOS is better than OSF/1 AD.

7: Conclusions:

(1) In SUNMOS, loading of data from memory to the processor is much faster than under OSF. This is clearly shown by the better performance of the first iteration of the 1-D FFT kernel.

(2) In SUNMOS, one can not use more than 24 MB of memory per node although SUNMOS needs only 250 KB of memory per node. This means that on a 32 MB per node Paragon about 7.5 MB memory is wasted due to the hardware problem.

(3) In SUNMOS, there is no paging effect, *i.e.*, performance of the first iteration and the second iteration are always the same. Under OSF, the dynamic allocation of memory enhances the performance of applications.

(4) In SUNMOS, there is no virtual memory. Even in the absence of the Paragon's hardware

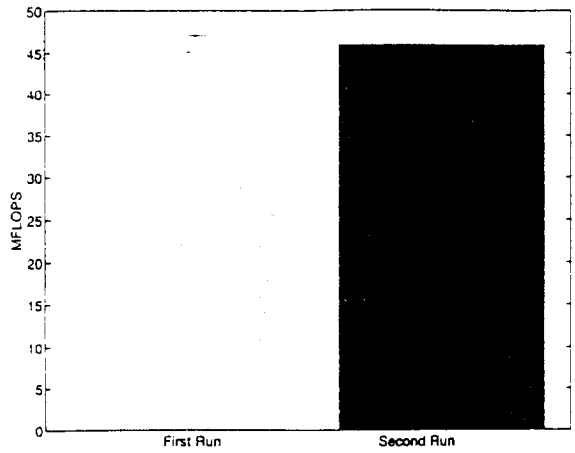

Figure 4(a): DGEMM on 1 node under OSF.

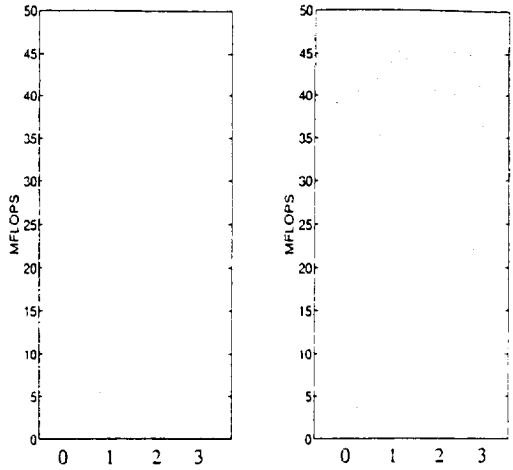

Figure 4(b):DGEMM on a 4 nodes under OSF - static versus dynamic memory allocation.

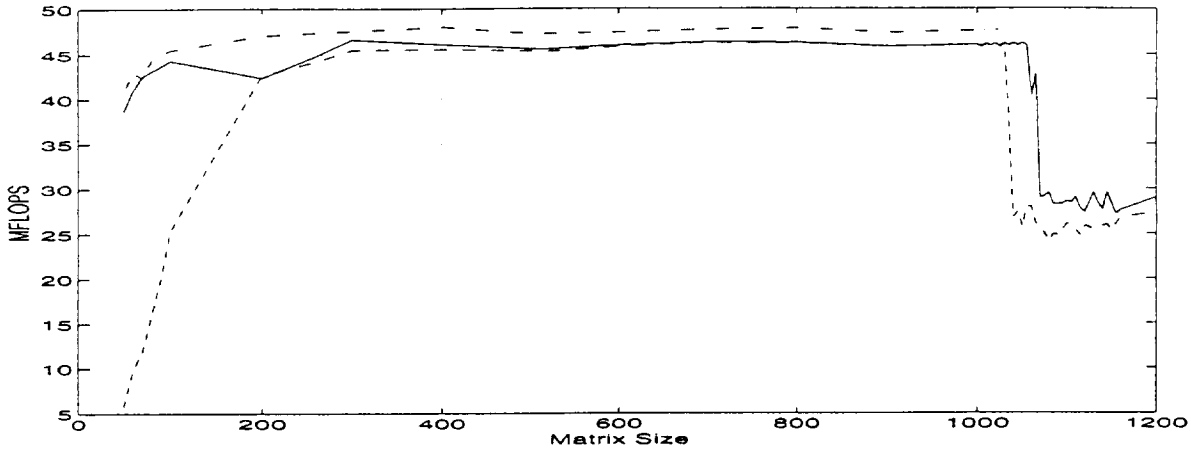

Figure 5: MFLOPS Vs. Size of the matrix, . _ . _ . _ SUNMOS;, - - - - 1st iteration (dynamic allocation), _____ 2nd iteration (dynamic allocation). Both 1st and 2nd iterations are under OSF.

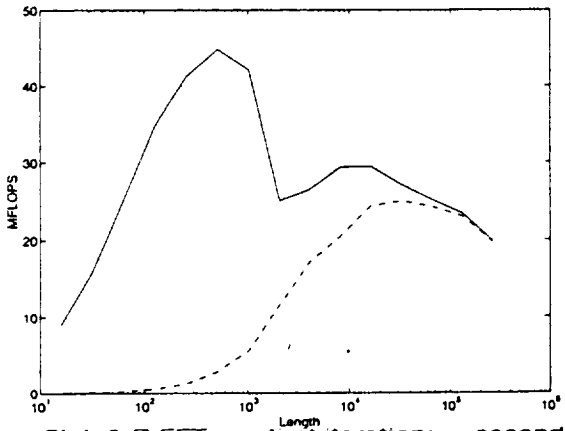

Figure 6(a): 1-D FFT, - - - first iteration; — second iteration; both are assembly codes under OSF.

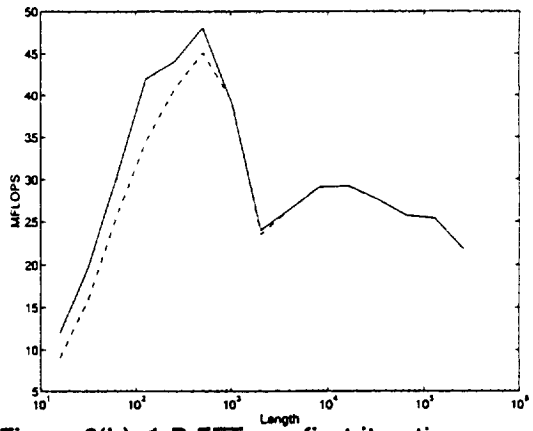

Figure 6(b): 1-D FFT, - - - first iteration; — second iteration; both are assembly coded under SUNMOS.

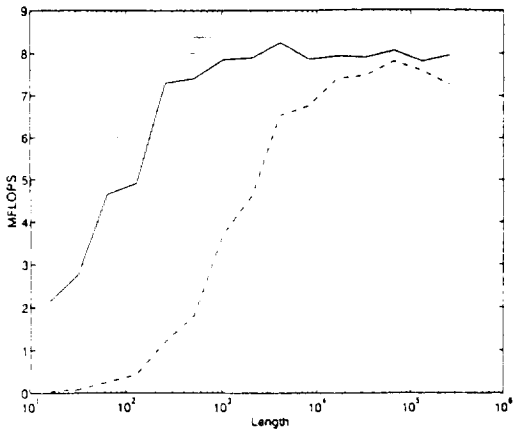

Figure 7(a): 1-D FFT, - - - first iteration, - second iteration; both are Fortran under OSF.

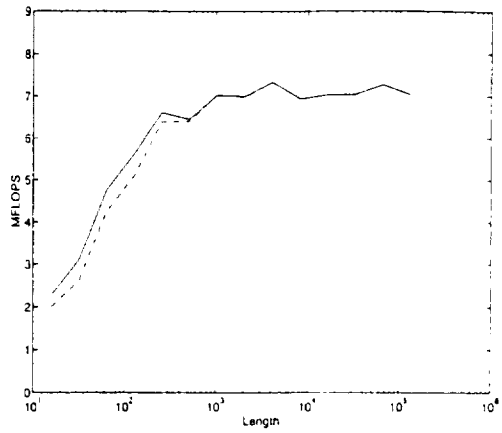

Figure 7(b): 1-D FFT, - - - first iteration, - second iteration; both are Fortran under OSF.

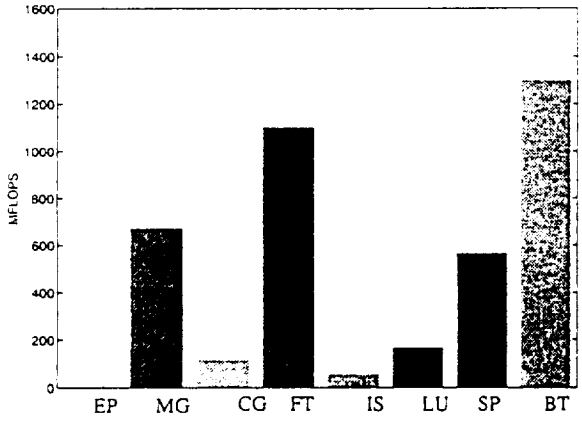

Figure 8 (a): Performance of NPB under OSF for 128 nodes.

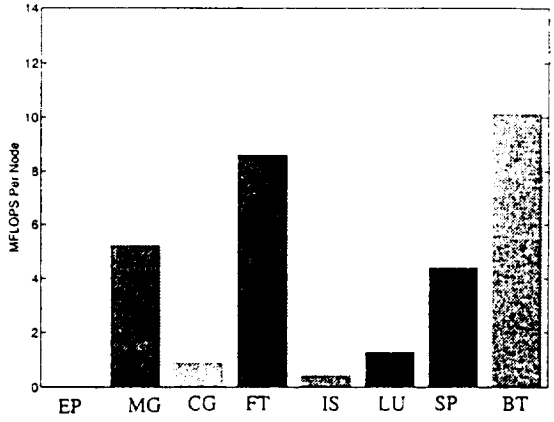

Figure 8 (b): Performance of NPB under OSF on one node.

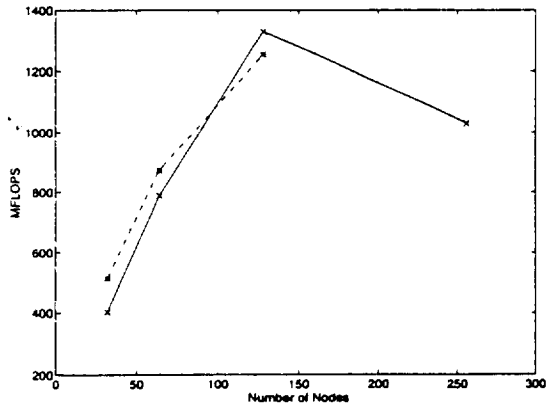

Figure 9(a): MFLOPS Vs. number of nodes.
_____ denotes results under OSF,
- - - - denotes results under SUNMOS.

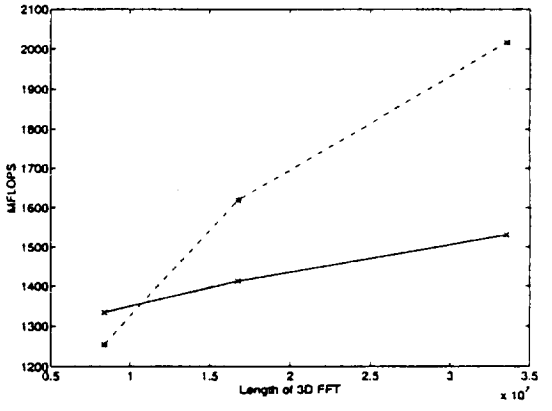

Figure 9(b): MFLOPS Vs. length of 3-D FFT.
_____ denotes results under OSF,
- - - -denotes results under SUNMOS.

problem, the maximum memory per node that can be utilized by the application will never exceed 31.5 MB. Currently, using 24 MB per node hangs or crashes the system.

(5) In SUNMOS, to run the application successfully either for a fixed problem size and varying number of nodes or for a fixed number of nodes and varying size problem, one has to adjust some or all of the following: (a) the size of the communication buffer, (b) the size of the heap, (c) the size of the stack. Choosing any one of them incorrectly either hangs or crashes the system or causes the system to be unusable until it is rebooted.

(6) In SUNMOS, the performance of the applications is generally (but not always) better than that under OSF.

(7) In SUNMOS, some enhancement of the applications' performance is attributed to the ability of SUNMOS to allocate the user's partition as close to a "square" as possible.

(8) On 128 nodes of the Paragon, only two NPB codes (FT and BT) exhibit performance of more than 1 GFLOPS. This high performance is due to use of optimized assembly routines for the 1-DFFT in FT benchmark and block tridiagonal solver in BT benchmark.

(9) For most of the NPB codes the performance is less than 5 MFLOPS per node, except for that of EP, FT and BT. The high performance of 12 MFLOPS for EP is due to it's having almost no communication. Message passing is used only to collect the results from different nodes. The use of optimized assembly routines (1-D FFT in benchmark FT and block tridiagonal solver in BT) enhance the performance.

(10) In OSF, the 3-D FFT kernel does not scale as the number of nodes are increased from 32 to 256.

8: Acknowledgment:

The authors wish to thank Thanh Phung of Intel SSD for useful discussions.

* This work is supported through NASA contract NAS 2-12961.

[1] E. Anderson et al., *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[2] *Overview of the i860TM XP Supercomputing Microprocessor*, 1991, Intel Corporation.

[3] D. Bailey et al., eds, *The NAS Parallel Benchmarks*, Technical Report RNR-91-02, NAS Ames Research Center, Moffet Field, California, 1991.

[4] D. Bailey et al., eds, The NAS Parallel Benchmark Results 394,Technical Report RNR-94-06, NAS Ames Research Center, Moffet Field, California, 1994.

[5] D. Bailey et al., The NAS Parallel Benchmark Results, IEEE Parallel & Distributed Technology, 43-51, February 1993.

[6] R. Zajcew et al. "An OSF/1 Unix for Massively Parallel Multicomputers", in *Proceedings of the 1993 Winter USENIX Conference*, January 1993, pp. 37-55.

[7] K. Loeppere, "OSF Mach: Kernel Principles", Open Software Foundation and Carnegie Mellon University, February 1993.

[8] B. Maccabe, K. S. McCurley and R. Rissen, SUNMOS for Intel Paragon: A Brief user Guide, November 29, 1993.

[9] K. S. McCurley, Intel NX compatibility under SUNMOS, Sandia National Laboratories, Albuquerque, Technical Report No. SAND 93-2618, Nov. 29, 1993.

[10] Paragon OSF/1, User Guide, Intel Corporation, 1994.

[11] *iPSC/860 ProSolver-SES Manual*, May, 1991, Intel Corporation.

[12] *iPSC/860 ProSolver-DES Manual*, March 1992, Intel Corporation.

[13] S. Saini and H.D. Simon, "Performance of BLAS and NAS Parallel Benchmarks on NAS Intel Paragon XP/S-15" in Proceedings of Intel Supercomputing User's Group Meeting, Oct. 3-6, 1993, St. Louis, Missouri, USA.

[14] D.H. Bailey and J. Barton, "The NAS Kernel Benchmarks Program", Report Number 86711, August 1985, NASA Ames Research Center.

[15] S. Saini and H.D. Simon, "Performance of BLAS 1, 2 on NAS Intel Paragon XP/S-15" in Proceedings of Scalable Parallel Libraries Conference organized by IEEE, Oct. 6-9, 1993, Mississippi State University, Starkville, Mississippi, USA.

[16] S. Saini and H.D. Simon, "Enhancing Applications Performance on Intel Paragon through Dynamic Memory Allocation", Report RNR-93-017, November 1993, NAS Systems Division, NASA Ames Research Center, Moffett Field, California 94035, USA.

[17] CLASSPACK, Basic Math Library User's Guide, Kuck & Associates, Release 1.3, 1992.